TABLE 1

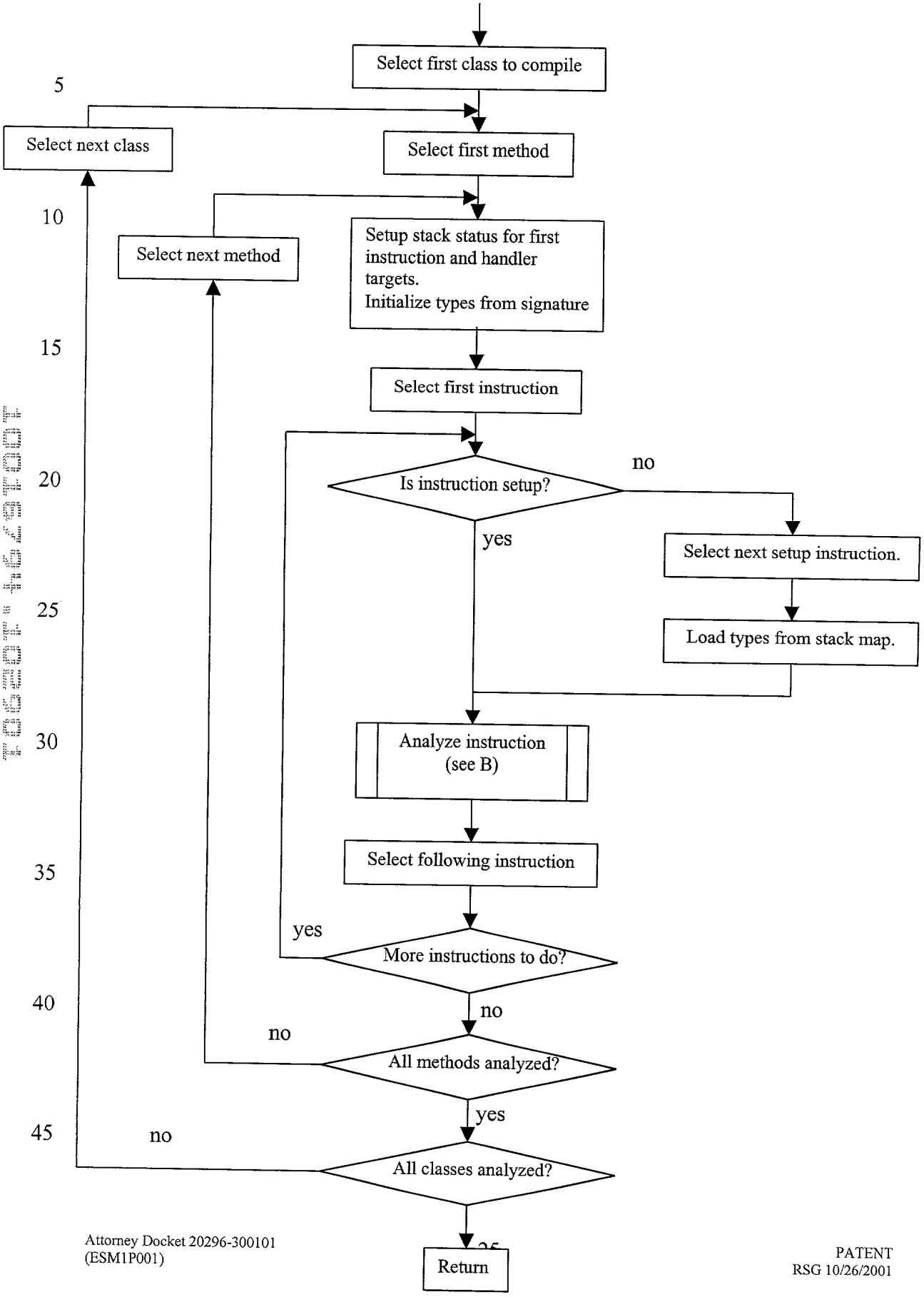
Combined Compilation and Verification of JAVA bytecode

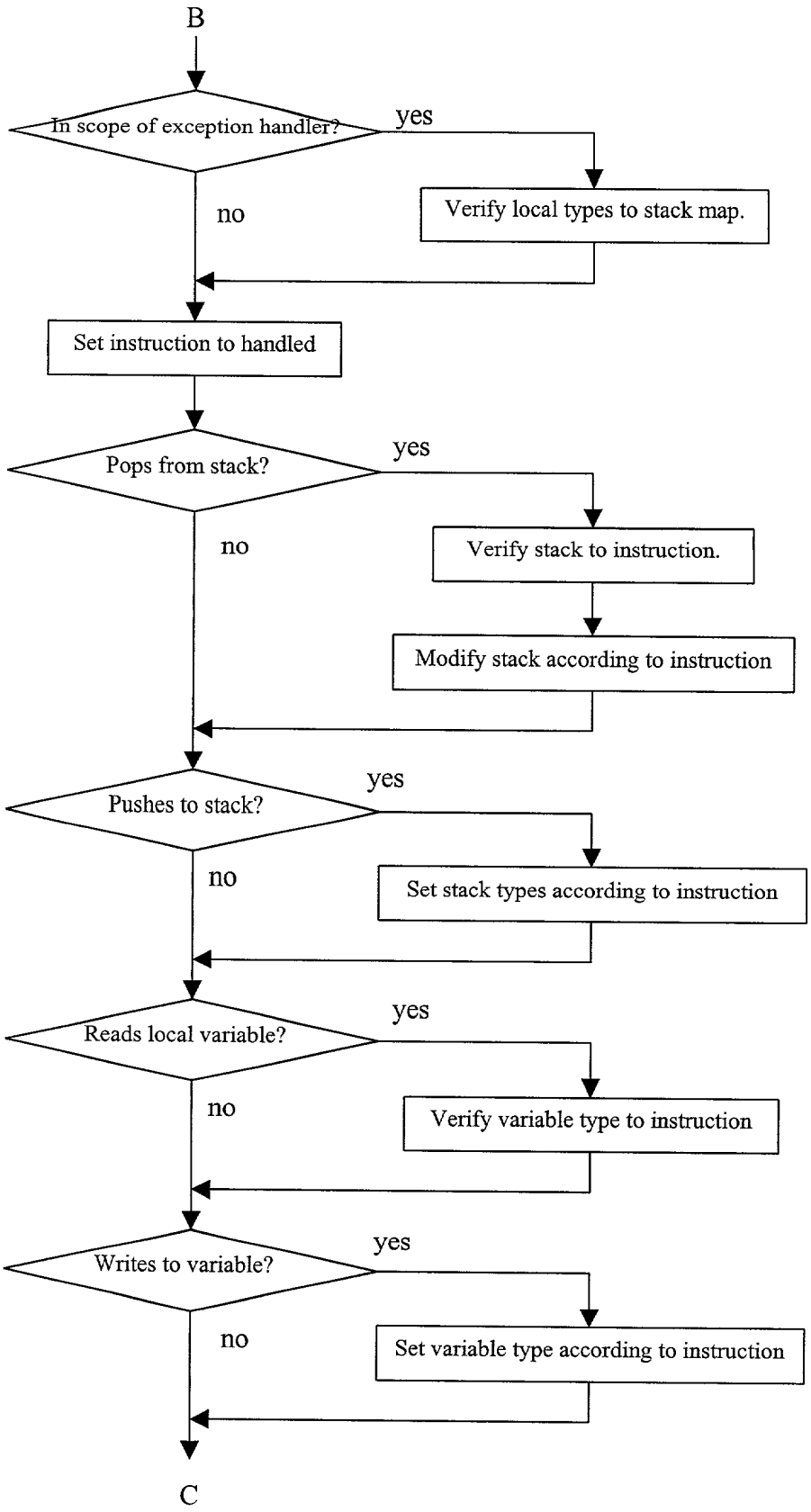
```
Object Class File containing bytecode instructions received.
For each Class File to be analyzed
{
    For each method in the class containing bytecode instructions
    {
        Create storage for each instruction to store stack status and marks.
        Create storage to store actual types of stack values and local variables.
        Initialize stack status of first instruction to empty.
        Initialize stack status of exception handler target instructions to contain the given
            exception object.
        Set marks of first instruction and handler target instructions to 'setup'.
        Set all other marks to 'none'.
        Initialize actual local variable types from method signature.
        Set the first instruction of the bytecode to be the actual instruction.
        Do until there are no more instructions marked as 'setup'
        {
            If actual instruction is not marked as 'setup'    /* New basic block */
            {
                Select next instruction in the bytecode marked as 'setup' as the actual
                    instruction.
                Load actual stack and local variable types from the stack map in bytecode
                    belonging to the actual instruction.
            }
            If the actual instruction is in the scope of an exception handler
            {
                Verify compatibility between actual local variable types and stack map for
                    the exception handler entry in bytecode.
            }
            Set the mark of selected instruction to 'handled'.
            Copy stack status of actual instruction to new stack status.
            If the actual instruction pops one or more values from the stack
            {
```

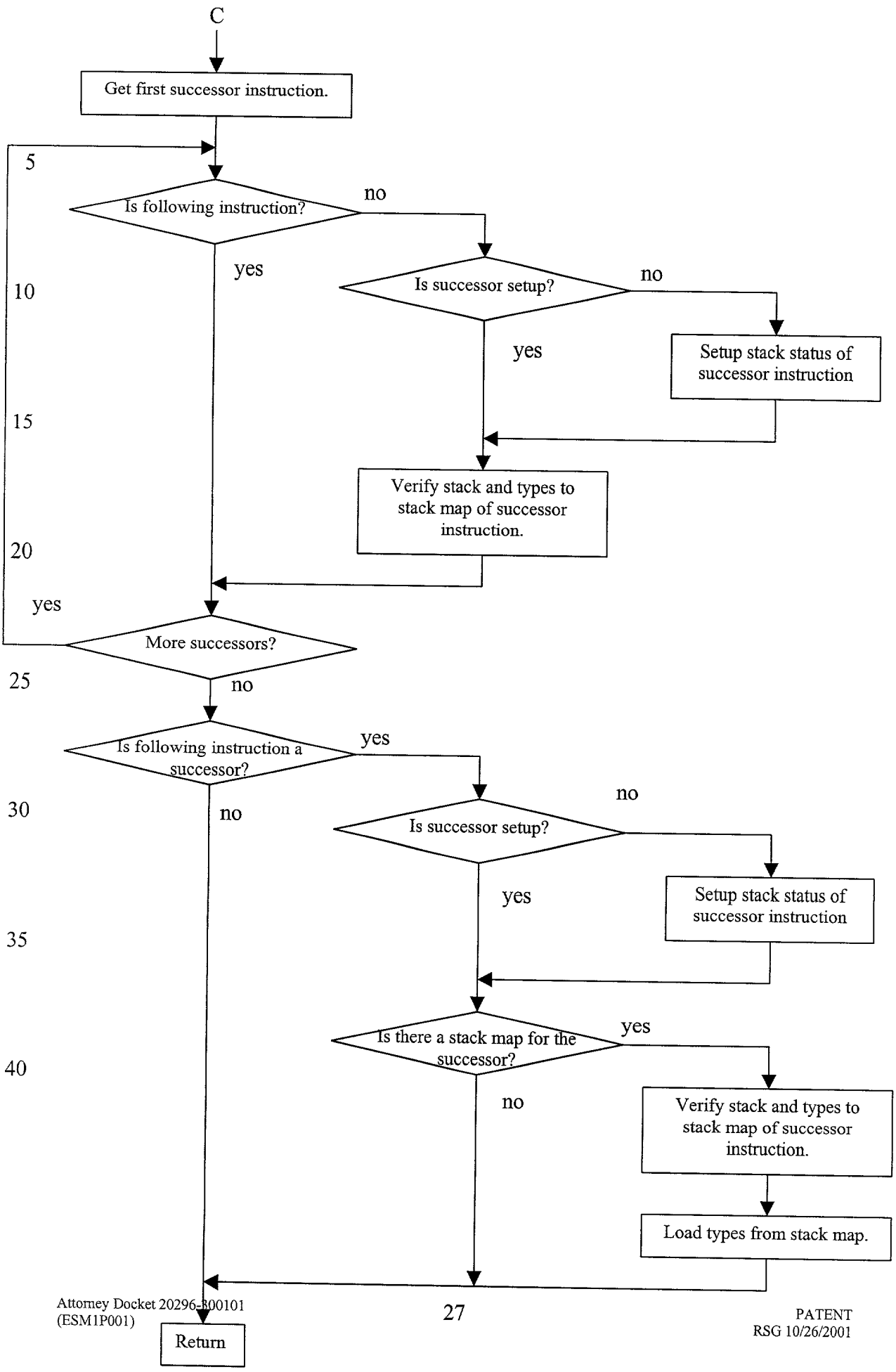
```
        Verify compatibility between the stack status and types and the values
            expected by the instruction.
        Modify new stack status according to the instruction.
    }
    If the actual instruction pushes one or more values to the stack
    {
        Modify new stack status according to the instruction.
        Set new actual stack types according to the instruction.
    }
    If the actual instruction reads a local variable
    {
        Verify compatibility between actual local variable types and the
            instruction.
    }
    If the actual instruction writes to a local variable
    {
        Modify actual local variable types according to the instruction.
    }
    For all successor instructions except the one immediately following the
        actual instruction
    {
        If the successor instruction is marked as 'none'
        {
            Initialize the stack status of the successor instruction to the new stack
                status.
            Mark successor instruction as 'setup'.
        }
        Verify compatibility between new stack status and stack map for the
            successor instruction in the bytecode.
        Verify compatibility between actual stack and local variable types and stack
            map for the successor instruction in the bytecode.
    }
    If the instruction immediately following the actual instruction is a successor of
        the actual instruction
    {
        If following instructions is marked as 'none'
        {
```

```
                Initialize the stack status of following instruction to the new stack status.
                Mark following instruction as 'setup'.
            }
            If there is a stack map in the bytecode for the following instruction
            {
                Verify compatibility between new stack status and the stack map.
                Verify compatibility between actual stack and local variable types and the
                    stack map.
                Load the actual types from the stack map.
            }
        }
        Change the actual instruction to the immediately following instruction.
    } /* Do until */
  } /* For each method */
} /* For each class file */
```

TABLE 2




B
In scope of exception handler?
yes
no
Verify local types to stack map.
Set instruction to handled
Pops from stack?
yes
no
Verify stack to instruction.
Modify stack according to instruction
Pushes to stack?
yes
no
Set stack types according to instruction
Reads local variable?
yes
no
Verify variable type to instruction
Writes to variable?
yes
no
Set variable type according to instruction
C


C
Get first successor instruction.
Is following instruction?
no
yes
Is successor setup?
no
yes
Setup stack status of successor instruction
Verify stack and types to stack map of successor instruction.
More successors?
yes
no
Is following instruction a successor?
yes
no
Is successor setup?
no
yes
Setup stack status of successor instruction
Is there a stack map for the successor?
yes
no
Verify stack and types to stack map of successor instruction.
Load types from stack map.
Return

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)_____ of _____ were not present for scanning. (Document title)

☐ Page(s)_____ of _____ were not present for scanning. (Document title)

☑ ***Scanned copy is best available.*** There are 21 pages of specification and 6 pages of table.